Vol. 00 no. 00 2010
Pages 1–2

TRStalker: an Efficient Heuristic for Finding Fuzzy Tandem Repeats - Supplementary materials

Marco Pellegrini [1], M. Elena Renda [1] and Alessio Vecchio [2]
[1]CNR, Istituto di Informatica e Telematica, Via Moruzzi 1, 56124 Pisa (Italy).
[2] Univ. di Pisa, Dip. Ingegneria dell'Informazione, Via Diotisalvi 2, 56122 Pisa (Italy).

Received on ; revised on ; accepted on

Associate Editor:

1 PSEUDOCODE OF TRSTALKER

The high level pseudocode of the TRStalker algorithm is reported in Figure 1, where each function embodies one of the functionalities described in the main paper.
At step 2., the function $block(Y)$ splits the input sequence Y into n blocks Y_j, $1 \leq j \leq n$, of predefined length. For each block Y_j, at step 3. function $findGappedQGrams(Y_j)$ (see Section 3.5 in the paper) records for each occurrence of a gapped q-gram P^i in Y its distances K' to the next 5 occurrences (candidate periods) and its starting position i in Y. Furthermore, function $updateWeight(k,i)$ (step 6.) increments the weight $w(k)$ of periods $k \in K'$ by applying two weighting techniques: the anti-smear weighting and the multiplicity weighting (see Section 3.6 in the paper).
The function $posDensity(K)$ (step 9.) computes the density of probes that contribute to each candidate period (see Section 3.7 in the paper), and the cuts off for position with low density, returning those candidates with higher density. Function $getTopPeriods(\widetilde{K}, L)$ (step 10.) ranks the periods by *weighted frequency* and returns only the top L positions (for $L = 50$ in our experiments) in the set $\widetilde{K}_L$.
For each candidate pair (k, i), functions $getTR()$ (step 12.) and $verifyTR()$ (step 13.) compute a candidate TR and verify whether there is a tandem repeat t of period k starting in position i according to the definition of TR (NTR or Steiner-STR - see Section 3.2 in the paper), and if so t is added to the set T. Finally, for each candidate TR $t \in T$ the function $maximal()$ verifies whether t is included in a longer TR, and possibly remove t from T, while $minp()$ for TRs in the same position and length but different period maintain only the TR with shorter period. The procedure returns T as result. During the visualization phase, the elements of T can be listed according to different properties of the TR found (e.g. initial position, final position, repeating unit size, number of repetitions, total length, absolute divergence, mean divergence, etc..)

```
Let $Y$ be the input string:
 1.  $L = 50, K, T = \{\}$
 2.  for each $Y_j$ $block(Y)$
 3.      for each $P^i$ in $findGappedQGrams(Y_j)$
 4.          $K' = periods(P^i)$
 5.          for each $k$ in $K'$
 6.              $w(k) = updateWeight(k, i)$
 7.              if $!((k, i) \in K)$
 8.                  $K = K \cup (k, i)$
 9.  $\widetilde{K} = posDensity(K)$
10.  $\widetilde{K}_L = getTopPeriods(\widetilde{K}, L)$
11.  for each $(k, i)$ in $\widetilde{K}_L$
12.      $t = getTR(k, i)$
13.      if $verifyTR(t)$
14.          $T = T \cup t$
15.  for each $t \in T$
16.      if $(!maximal(t) \mid\mid !minp(t))$
17.          $T = T \setminus t$
18.  return $T$
```

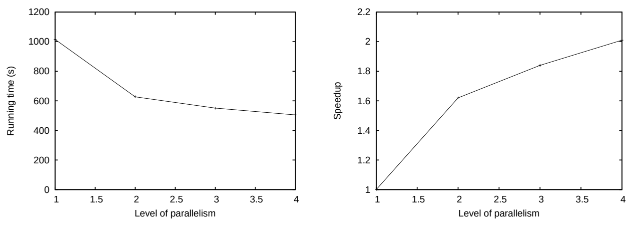
Fig. 1. TRStalker algorithm scheme.

2 RUNNING TIME OF TRSTALKER

The running time of the algorithm has been analyzed to evaluate the cost associated with the different activities. The evaluation has been performed by using a JVM profiler that measures the time spent within the methods implementing $i)$ the localization of gapped q-grams within the sequence, $ii)$ anti-smear weighting of distances, $iii)$ multiplicity weighting, $iv)$ computation of positional density, and $v)$ validation of TRs.

Localization of gapped q-grams	0.25%
Anti-smear weighting	18.49%
Multiplicity weighting	0.13%
Computation of positional density	2.42%
Validation	77.63%
Other	1.08%

Table 1. Percentages of time spent within the different activities.



(a) Running time (b) Speedup

Fig. 2. Running time and speedup achieved by parallel execution of the algorithm on a four cores CPU (Intel Core i7 860 - 2.8 GhZ) when analyzing 50 sequences of length 4000bp (each sequence contains an implanted Steiner-TR with motif length equal to 50bp and repeated eight times).

Table 1 reports the amount of time dedicated to the previous five activities (the values are the average over ten executions of the algorithm). The experimental evaluation has been carried out by using synthetic sequences of length 4000bp and containing an implanted Steiner-TR with motif length equal to 50bp and repeated eight times. The amount of substitutions, insertions and deletions is equal to 10% of the motif length each. The hardware used to run the experiment has the following characteristics: CPU Intel Core i7 860 - 2.8 GhZ, RAM 4 GB, Linux Operating System, Java HotSpot 64-bit Server VM 1.6.0.

The current implementation of TRStalker, despite being not yet fully optimized, includes a parallelized execution of the algorithm, that reduces the execution time by exploiting multi-core architectures of current CPUs. Parallelization is implemented at the level of q-gram shapes: for every shape a different thread of execution is used, then the results are merged. This rather simple technique provides the speedup depicted in Figure 2(b) and scales reasonably considering the number of cores of current CPUs (the absolute running time is shown in Figure 2(a)). For long sequences a further level of parallelization can be introduced by analyzing separately the blocks a sequence is divided into. This second technique could be useful to reduce the running time on architectures with higher degree of parallelism (such as clusters).